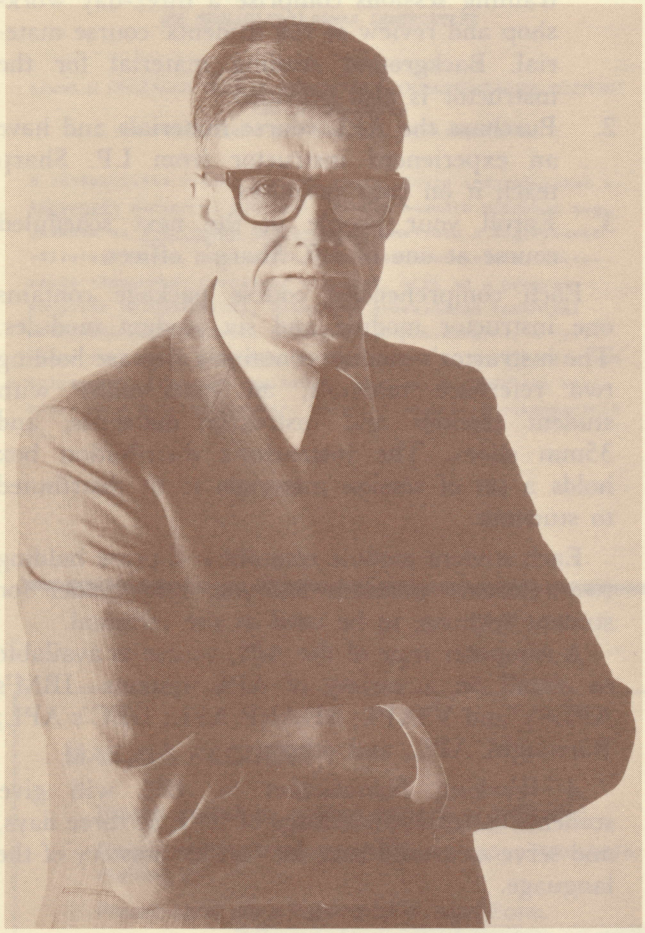
SEPTEMBER/OCTOBER 1981
Volume 9 / Number 5

I.P. Sharp newsletter



A fresh approach to teaching APL

"I have long been struck by the contrast between the success with which the adventurous learn APL by simply using it, and the frequent failure of lecture courses to communicate the simplicity and applicability of the language."
Kenneth E. Iverson

Ken Iverson, whose name has been synonymous with APL for over 20 years, has designed an introductory APL course. *A Working Introduction to APL* is an alternative to traditional lecture courses that have made the language unnecessarily difficult to understand. This course will provide the student with the essential ingredients for a "working" knowledge of APL. It is structured to allow rapid understanding of the language without any prior computer programming knowledge.

CONTENTS

1 A Fresh Approach to Teaching APL

Application Software
3 Project Management
4 Workspace of the Month
4 Library Update

Meetings
6 Winnipeg Week
6 APL Forum in Vienna

Bulletin Board
5 Hong Kong
6 Vienna, Halifax, Budapest

Technical Supplement 34
T1 Application of Complex Numbers to Sampled Data Systems
T3 Comment
T4 Statement Separators
T5 NY SIGAPL—A Report
T6 A Model for the Extension of Monadic Thorn
T9 Shared Variables 2
T12 Note on Part 1 of Shared Variables.

Your organization can purchase this course for use in your in-house training centre, or enroll students in the course at one of I.P. Sharp's branch offices.

The three-day introductory APL course will serve as an ideal first course for:

1. The professional who wishes to use computers in day-to-day work in his own discipline.
2. Anyone wishing to become an APL programmer capable of designing and producing APL application packages for use by others.
3. People who think a computer would be useful in their present work, or in extending its scope.
4. The manager who directs the work of APL programmers, and realizes the importance of gaining some first-hand experience in a language that differs substantially from other programming languages.
5. Those who are now using an APL application written by someone else, and want to know more.

Course material is carefully structured to introduce the student to the essential elements of APL while prompting immediate exploration of the problem-solving capabilities intrinsic to the language. As he or she explores and learns, confidence and independence are achieved simultaneously.

Students work primarily at terminals with guidance from a qualified instructor; the summary lectures are few, brief and practical. They learn by experimentation and by observation of well-chosen examples—working at their own pace. They are led into serious use of the reference materials. Supplementary exercises ensure a natural continuation of the learning process for all students after the conclusion of the course.

Your organization can take advantage of this course in several ways:

1. Purchase the APL course materials and teach it on your own premises. I.P. Sharp will train a qualified APL programmer in the special instructional techniques for this course. One of your qualified APL programmers will be trained as a course instructor. The instructor training sessions comprise a three-day workshop and review of the students' course material. Background reading material for the instructor is also provided.
2. Purchase the APL course materials and have an experienced instructor from I.P. Sharp teach it on your premises.
3. Enroll your people in the next scheduled course at one of I.P. Sharp's offices.

Each comprehensive course package contains one instructor module and six student modules. The instructor's module contains a slipcase holding two reference manuals, an easel-binder with student sessions and instructor materials, and 35mm slides. The instructor's distribution box holds a set of session materials to be distributed to students.

Each student module contains a slipcase holding two reference manuals and one easel-binder for student sessions, to be used at the terminal.

A computer tape of the APL course is available to install on a variety of APL systems—IBM's APLSV and VSAPL; SHARP APL; DEC's APL; Burroughs APL; and others.

A Working Introduction to APL will give students essential knowledge of APL in three days, and serve as a solid base for further mastery of the language.

PROJECT MANAGEMENT

The Modern Approach with SNAP

Ken Chakahwata, London

We are often asked - when was SNAP released?

Well, it is rather difficult to say, since SNAP is one of those systems that truly evolved, starting about four years ago with simple utilities to enter project network data, perform basic critical path analysis, and produce very simple reports. This basic structure hasn't changed much since then; what has changed somewhat is our general approach to project management applications.

The requirement for critical path analysis facility was expressed by a couple of our customers, who were in fact using these simple facilities more or less as they were written. Our response to the very valuable feedback from these early users resulted in continuous embellishments and enhancements, some of which were major design changes.

It is interesting to note that a lot of early user feedback was apparently a result of previous bad experiences with other project management systems. The dominant criticism was that most systems of this nature are "too big, cumbersome, and inflexible" for the typical user application. At first this presented us with what looked like a dilemma. We had to continue enhancing our utilities to satisfy the ever changing requirements of our clients - but at the same time, we had to somehow avoid falling into the trap of developing "yet another monolith".

The answer came from experience gathered from more interactions with our clients. We began to realise that a project manager is in a lot of ways no different from any other manager. They deal with a diversity of disciplines. Their priorities, (hence requirements) change continually and unpredictably - in most cases due to external uncontrollable factors. What sets aside the project manager is that he normally works towards a more specific goal, under specific time and cost constraints. Success in the environment depends on the availability of flexible tools for evaluating past performance, then learning from this to control the current situation, and then plan for the future.

What resolved the apparent dilemma for us was the important observation that not all project managers' requirements translate into "frozen" facilities within a computer package. What was needed was a system of building blocks, or primitives, open and flexible enough to allow the project manager to combine them to address specific requirements.

SNAP is a set of these building blocks. To a large extent, the building block philosophy is rather trivial. More important was the identification of what primitives were required. Our extensive experience with various types of projects, manufacturing, research and development, software, construction, plant relocation, etc., helped enormously.

SNAP has been used in this way rather heavily in the planning, coordination and control of multilocation projects that take advantage of the powerful file sharing control facilities within SHARP APL and of the I.P. Sharp international communications network. We have been able to handle projects reflecting a variety of organisational structures - hierarchical, matrix, and mixtures.

So, while SNAP has been used for some time, it has reached a milestone where we consider it officially released. The rate of evolution has slowed down, the product is stable and robust, and a new manual is available.

To the new SNAP user we can demonstrate how quickly you can get from a rough back-of-an-envelope network draft to quite a detailed plan, covering both time and resource aspects of your project; and more importantly, how easy it is to make and examine consequences of changes so you can plan for them.

The established SNAP user is already aware of the power of SNAP with SHARP APL. In most cases, the level of customisation reaches a stage where we cannot tell whether the basic software is SNAP or not.

For the modern project manager faced with the challenges of the 80's, but perhaps having only the old inadequate management tools at hand, we have an important message. Phone your nearest I.P. Sharp office now!

Workspace of the Month

SORTING SMALL FILES QUICKLY AND EASILY

Brian G. Sharratt, Toronto

A workspace which is not "new", but which deserves to be brought to your attention, is 85 *APLSORT*. It provides software to sort files. Where appropriate, it makes use of the SHARP APL extension to ⍋ to more efficiently handle the sorting of alphabetics. It is a good alternative to 1 *FILESORT* since it is designed for interactive use; you will seldom need more than a minute to set the parameters and sort the file.

Because *APLSORT* runs as a T- or N-task, it is most suitable for a small file; for a larger file, batch *FILESORT* is much cheaper because the sorting is done in background and not by the APL interpreter. Of course, there will be a delay incurred in using such a batch sort facility.

APLSORT makes sorting easy by determining most of the constraints itself (e.g., data type, number of rows and columns). You provide only the input and output file names, the sorting sequence (ascending or descending for each column, and the column order), and the collating sequence for character files.

A recent enhancement now allows you to optionally specify the blocking factor or the number of rows in each component of the output file.

For complete informaiton on how to use *APLSORT*, load the workspace and type *DESCRIBE*.

APPLICATION LIBRARY UPDATE

2 *DIRECTDEF*	→→	has been superseded by 1 *DIRECTDEF* and is now in library 499.
24 *COMPENS* 25 *CMATRIX*	→→	The introduction of complex numbers in APL eliminates the need for these workspaces.

WINNIPEG WEEK

As a result of an invitation to present a paper at the Sixteenth Annual Actuarial Research Conference (sponsored by the Committee on Research of the Society of Actuaries), **Ken Iverson**, along with **Jerry Cudeck** and **Jim Spurgeon** of our Corporate Actuarial Department, enjoyed a week in wonderful Winnipeg.

Ken used the opportunity to present his talk on APL myths to an overflow crowd at a luncheon meeting arranged by I.P. Sharp, and to discuss the use of APL in teaching at a meeting of faculty organized by Professor Amir Bukhari (Department of Actuarial and Business Mathematics) at the University of Manitoba. Following the conference, Jerry presented his two-day seminar entitled 'Advanced APL Coding Techniques for Actuarial Applications' to a large group drawn from insurance companies and actuarial consulting firms in the Winnipeg and Regina areas, as well as a number of faculty members of the University of Manitoba.

The paper, 'The Contingent Benefits of APL', emphasized the use of APL as an analytical tool in actuarial work. It will be published in ARCH (Actuarial Research Clearing House). Preprints are available from Jim Spurgeon.

Because of the enthusiastic response to Jerry's seminar in Winnipeg, we will offer a repeat in Toronto on December 7 and 8. Those interested in attending this session should contact Jerry Cudeck or Jim Spurgeon.

Toronto APL Special Interest Group

DISCUSS APL81—6pm October 28/ December 2

All of you who attended the San Francisco conference (and those who could not), will have the opportunity to discuss some of results, at the Toronto Board of Trade (1st Canadian Place, Adelaide St., 3rd fl. private elevator).

To make a reservation, please leave your name, phone and company name for Dan King at 979-2035.

APL FORUM IN VIENNA

Gottfried Bach, Vienna

The APL User Forum normally takes place about every two to three months, and is meant to be a somewhat informal gathering of a small group of people interested in APL. It is not restricted to SHARP APL users.

Meetings in the past have featured topics such as Event Trapping, Graphics, Highlights of SHARP APL, An Evening with Paul Berry, and Complex Numbers.

The September meeting was a special occasion: Ian Sharp happened to be in Vienna, and the new release of SHARP APL provided several interesting topics.

Almost thirty APL'ers attended the new forum at the Hotel Stefanie in Vienna. The program:

Growth and Future of APL, I.P. Sharp, I.P. Sharp Associates, Toronto
Complex Numbers in APL, H. Mach, Philips Vienna
Enclosed Arrays in APL, G. Bach, I.P. Sharp Associates Vienna
New Operators in APL, F. Vrabec, Eumig Vienna

Ian Sharp's talk showed very nicely the development of APL since its beginning, and provided the opportunity for a number of interesting questions such as "Where is APL going?", "What will the role of minis and micros be in the future?", "Data security in the SHARP APL timesharing system", and so on.

The technical people enjoyed Mr. Helmut Mach's presentation of complex numbers. He gave a brief introduction and showed some interesting applications in his field. Please turn to the article "Applications of Complex Numbers to Sampled Data Systems" in the technical supplement of this newsletter.

Enclosed arrays were received with considerable interest. It was quite clear that most users could immediately think of many applications where programming might become simpler by using enclosed arrays.

Unfortunately, Mr. Vrabec was unable to present his paper himself, so this was done by Gottfried Bach. Time was getting short, though, and we decided to have a special user forum just on "New Operators in APL" toward the end of September.

HONG KONG
On The Occasion Of The Opening....

James Sinclair, Hong Kong

There are many theories as to why Hong Kong is among the last British Colonies; the most interesting is that the weather is to blame. The British extract extraordinary conversational mileage from what is, after all, a rather dull climate back home. How then could they possibly relinquish Hong Kong with its obvious attractions? Not only is the summer unbearably hot and humid, (it is fashionable to complain of the heat), but there is a real bonanza typhoons!

James Sinclair

The Grand Opening Reception of the I.P. Sharp Associates Hong Kong office was very nearly cancelled by one such bonanza. Mounting levels of typhoon warning were hoisted over the preceding days. A ghoulish excitement prevailed as, for the second time this summer, Hong Kong battened down the hatches. But it was not to be: Typhoon Clara changed her mind, and veered off into southern China.

As it turned out the Reception was an acclaimed success. About 150 people, mostly from the banking, insurance and shipping sectors of the community, attended the event which was

co-hosted by the Commission for Canada. It was particularly pleasing to see a large number of guests from local Hong Kong corporations and government.

The office is in the Admiralty Centre, a new office/shopping complex in the Central business district. The centre incorporates an Underground station and a bus station and is about three minutes walk from the 'Star Ferry'. The office itself includes a classroom of a design which will be particularly suitable for the new 'Working Introduction to APL' course.

The *South China Morning Post* heralded the opening with an article entitled 'Computer Giant comes to Hong Kong' on the front page of the Business News section. The *Hong Kong Standard* announced, in their lead article under 'Company News', the opening of the fifty-seventh I.P. Sharp Office. Two different Chinese-language papers have noted our arrival as well, (unfortunately adopting different transcriptions of the company name, underlining the need to settle on a satisfactory translation).

Hong Kong is a vibrant and exciting city: to the foreigner it affords a curious mixture of East and West. Nine-tenths of the territory theoretically will be returned to China in less than sixteen years—it has been said that Hong Kong is "a borrowed place on borrowed time". However, unless there is a major turnabout in China, it is extremely unlikely that China would wish to change the status quo, which is profitable to both sides. In the mean time, Britain continues to pay the HK$5,000 per annum rent, which Peking graciously accepts, and SHARP APL services will continue to be available throughout the territory.

But a speck on the map, Hong Kong is among the top 18 trading nations and has the world's third busiest port. Business here is changing: the days of cheap labour have now departed these shores forever (wage costs are rising very fast) and never was productivity improvement so important. Programming staff is extremely scarce. Hong Kong will respond to SHARP APL, because SHARP APL can address these problems, both at the management and at the computer department level.

VIENNA

Christian Defregger (mailbox code *CDE*) joined the Vienna office on July 1st, 1981. He has a technical background and considerable experience with mini and microcomputers. Now that he has discovered the power of APL, he is very keen on developing large systems in APL.

Christian Defregger

Don Howson

HALIFAX

Don Howson moved to Halifax in late August to open our thirteenth Canadian office. A native Montrealer, Don has worked in the Ottawa office since graduating from Carleton University (Mathematics and Commerce) in 1977.

The Halifax office will be taking shape during the month of September, and will be providing both highspeed print service, and GRAPHREQ, the graphics output request facility, for customers in Atlantic Canada by mid-October.

The address of the office is:

I.P. Sharp Associates Limited,
Suite 706 Cogswell Tower,
2000 Barrington Street,
Halifax, Nova Scotia B3J 3K1
(902) 423-6251

BULLETIN BOARD

SHARP APL IN BUDAPEST

The following is an extract from "AZ I.P. SHARP NEMZETKöZI SZÁMITÓGÉPES HÁLÓZAT ÉS SZOLGÁLTÁTA SÁNAK ISMERTETéSE, (printed in Budapest, April 1981).

ORGASZOS MÜSZAKI KÖNYVTÁR ÉS DOKUMENTÁCIÓS KÖZPONT

AZ I.P. SHARP NEMZETKÖZI SZÁMITÓGÉPES HÁLÓZAT

ÉS SZOLGÁLTATÁSÁNAK ISMERTETÉSE

Készült az ORSZÁGOS MÜSZAKI KÖNYVTÁR ÉS DOKUMENTÁCIÓS KÖZPONT
házi sokszorositójában
A/4 méretben, 100 példányban, 35 oldal terjedelemben

A tájékoztatás széles körü és gyors. Rövid információkat a képernyőn keresztül közölnek. Minden második hónapban megjelenik az I.P. Sharp Tájékoztató, amelyben a legfrissebb vállalati hirek, rendszerfejlesztések szerepelnek. Változatos témakörben tartalmaz cikkeket, hirt ad a programkönyvtár változásairól, általános érdeklődésü technikai problémák megoldásáról, előzetes elképzelésekről, a nyilvános vagy közös adatbázis változásairól és a különféle tanfolyamokról. Hasznos segitséget nyujthat a SHARP APL Kézikönyv és a programkönyvtár, valamint adatbázis dokumentáció is.

A SHARP APL Kézikönyv a helyi kirendeltségeken keresztül beszerezhető. A 350 oldalas tájékoztató részletesen tartalmazza az APL programozási nyelvet, leirja a Sharp rendszer szervezeti felépitését és megadja az összes APL egység és rendszerfüggvény definicióját. Praktikus illusztrációk és indexek egészitik ki.

A programkönyvtár és adatbázis dokumentáció két formában érhető el. Korábban mindent meg lehetett kapni közvetlenül a terminálról /on-line formában/. A könyvtárállomány növekedésével felhasználói kézikönyvet készitettek, ami részletesen leirja a programkönyvtár tartalmát és használatát. A kézikönyv mellett Könyvtári Katalógus is ren-

UPDATE

MAILING REQUEST

- ☐ Please amend my mailing address as indicated.
- ☐ Please add the following name(s) to your Newsletter mailing list.
- ☐ Please send me a Publications Order Form.
- ☐ Please add my name to the Energy Newsletter mailing list.
- ☐ Please add my name to the Aviation Newsletter mailing list.
- ☐ Please add my name to the Financial Newsletter mailing list.
- ☐ Please send me information about your courses in

(city) _____

Name: _____

Title: _____

Co.: _____

Address: _____

_____

_____

Telephone: _____

The Newsletter is a regular publication of I.P. Sharp Associates. Contributions and comments are welcome and should be addressed to: I.P. Sharp Newsletter, 145 King Street West, Toronto, Canada M5H 1J8.

Jeanne Gershater, *Editor*
Mary Kopfensteiner, *Circulation*

Printed in Canada
October 1981
ISSN 0226 854X

I.P. Sharp Associates **Head Office:** 145 King Street West, Toronto, Canada M5H 1J8 (416) 364-5361

International Offices

Aberdeen
I.P. Sharp Associates Limited
5 Bon Accord Crescent
Aberdeen AB 12DH
Scotland
(0224) 25298

Amsterdam
Intersystems B.V.
Kabelweg 47
1014 BA Amsterdam
The Netherlands
(020) 86 80 11
Telex: 18795 ITS NL

Atlanta
I.P. Sharp Associates, Inc.
1210 S. Omni International
Atlanta, Georgia 30335
(404) 586-9600

Boston
I.P. Sharp Associates, Inc.
Suite 415
148 State Street
Boston, Massachusetts 02109
(617) 523-2506

Bristol
I.P. Sharp Associates Limited
5 Whiteladies Rd., Clifton
Bristol BS8 1NN
(0272) 314-486

Brussels
I.P. Sharp Europe S.A.
Avenue du General de
Gaulle, 39
1050 Bruxelles
Belgique
(02) 649 99 77

Calgary
I.P. Sharp Association Limited
550 BowValley Sq. 4
250-6th Ave. S.W.
Calgary, Alberta T2P 3H7, Canada
(403) 265-7730

Canberra
I.P. Sharp Associates Limited
16 National Circuit
Barton, ACT 2600
Australia
(062) 73-3700

Chicago
I.P. Sharp Associates, Inc.
2 North Riverside Plaza
Suite 1736
Chicago, Illinois 60606
(312) 648-1730

Cleveland
I.P. Sharp Associates, Inc.
(216) 431-6861
(local call, switched through
to Rochester office.)

Copenhagen
I.P. Sharp ApS
Ostergade 24B
1100 Copenhagen K
Denmark
(01) 11 24 34

Coventry
I.P. Sharp Associates Limited
7th Floor B Block
Coventry Point, Market Way
Coventry, England CV1 1EA
(0203) 21486/7

Dallas
I.P. Sharp Associates, Inc.
Suite 1148, Campbell Centre
8350 Northcentral Expressway
Dallas, Texas 75206
(214) 369-1131

Denver
I.P. Sharp Associates, Inc.
Suite 416
5680 South Syracuse Circle
Englewood, Colorado 80111
(303) 741-4404

Dublin
c/o Gamma Data Systems
Limited
Dollard House
Wellington Quay
Dublin 2, Ireland
(01) 711 877

Düsseldorf
I.P. Sharp GmbH
Leostrasse 62A
4000 Düsseldorf 11
West Germany
(0211) 57 50 16

Edmonton
I.P. Sharp Associates Limited
Suite 2358, Principal Plaza
10303 Jasper Avenue
Edmonton, Alberta T5J 3N6
(403) 428-6744

Gloucester
I.P. Sharp Associates Limited
29 Northgate Street
Gloucester, England GL1 2AN
(0452) 28106

Hamilton
I.P. Sharp Associates Limited
14 Hess South, Hess Village
Hamilton, Ontario L8P 3M9
(416) 527-3801

Halifax
I.P. Sharp Associates Limited
Cogswell Tower
Suite 706, 2000 Barrington Street
Halifax, Nova Scotia
B3J 3K1, Canada
(902) 423-6251

Hong Kong
I.P. Sharp Associates (HK) Limited
Suite 606, Tower 1
Admiralty Centre, Hong Kong
5-294341

Houston
I.P. Sharp Associates, Inc.
Suite 375, One Corporate Square
2600 Southwest Freeway
Houston, Texas 77098
(713) 526-5275

Kitchener/Waterloo
I.P. Sharp Associates Limited
3 Menno St.
Waterloo, Ont. N2L 2A4
(519) 884-5420

London, Canada
I.P. Sharp Associates Limited
Suite 510, 220 Dundas Street
London, Ontario N6A 1H3
(519) 434-2426

London, England
(European Headquarters)
I.P. Sharp Associates Limited
132 Buckingham Palace Road
London SW1W 9SA, England
(01) 730-0361
Telex: 8954178 SHARP G

Los Angeles
Suite 1230, 1801 Century Park
East
Los Angeles, Ca. 90067
(213) 277-3878

Madrid
I.P. Sharp Associates Ltd.
Serrano 23, Piso 8
Madrid - 1, Spain
(91) 276 70 54

Manchester
I.P. Sharp Associates Limited
Paul House
89-91 Buttermarket Street
Warrington, Cheshire
England WA1 2NL
(0925) 50413/4

Melbourne
I.P. Sharp Associates Pty. Ltd.
520 Collins St., 13th Floor
Melbourne 3000
Victoria, Australia
(03) 614-1766

Mexico City
Teleinformatica de Mexico S.A.
(Agent)
Mail to:
Arenal N 40, Chimalistac
Mexico 20 D.F., Mexico
(905) 550-8033

Miami
I.P. Sharp Associates, Inc.
Suite 240
15327 N.W. 60th Avenue
Miami Lakes, Florida 33014
(305) 556-0577

Milan
I.P. Sharp Srl (agent I.S.I.)
Via Eustachi 11
20129 Milan, Italy
(02) 271-6541

Montreal
I.P. Sharp Associates Limited
Suite 1610
555 Dorchester Boulevard W.
Montreal, Quebec H2Z 1B1
(514) 866-4981

New York City
I.P. Sharp Associates, Inc.
Mail to: Suite 2004
200 Park Avenue
New York, N.Y. 10169
(212) 557-1200

Newport Beach
I.P. Sharp Associates, Inc.
Suite 1135
610 Newport Center Drive
Newport Beach, Ca. 92660
(714) 644-5112

Oslo
I.P. Sharp A/S
Dronningens gate 34
Mail to: P.Boks 486 Sentrum
OSLO 1, Norway
(02) 41 17 04

Ottawa
I.P. Sharp Associates Limited
Suite 600, 265 Carling Ave.
Ottawa, Ontario K1S 2E1
(613) 236-9942

Palo Alto
I.P. Sharp Associates, Inc.
Suite 201, 220 California Ave.
Palo Alto, Ca. 94306
(415) 327-1700

Paris
I.P. Sharp Sarl
Tour Neptune, Cedex 20
20 Place de Seine
92086 Paris-la-defense
France
(1) 773 57 77

Philadelphia
I.P. Sharp Associates, Inc.
Suite 604, 437 Chestnut St.
Philadelphia, Pa. 19106
(215) 925-8010

Phoenix
I.P. Sharp Associates, Inc.
Suite 503
3033 N. Central Avenue
Phoenix, Arizona 85012
(602) 264-6819

Rochester
(United States Headquarters)
I.P. Sharp Associates, Inc.
(United States Headquarters)
1200 First Federal Plaza
Rochester, N.Y. 14614
(716) 546-7270
Telex: 0097 8473
0097 8474

San Francisco
I.P. Sharp Associates, Inc.
Suite C-415, 900 North Point St.
San Francisco, Ca. 94109
(415) 673-4930

San Jose
I.P. Sharp Associates, Inc.
3028A Scott Boulevard
Santa Clara, California 95050
(408) 727-9446

Saskatoon
I.P. Sharp Associates Limited
Suite 303, Financial Bldg.
230-22nd St. E. Saskatoon,
Saskatchewan S7K 0E9
(306) 664-4480

Seattle
I.P. Sharp Associates Inc.
Suite 223, Executive Plaza East
12835 Bellevue Redmond Road
Bellevue, Washington 98005,
(206) 453-1661

Singapore (Far East H.Q.)
I.P. Sharp Associates (S) Pte. Ltd.
Suite 1501, CPF Building
79 Robinson Rd.
Singapore 0106
Republic of Singapore
2230211

Stockholm
I.P. Sharp AB
Kungsgatan 65
S111 22 Stockholm, Sweden
(08) 21 10 19

Stuttgart/Boeblingen
I.P. Sharp GmbH
Schafgasse 3
7030 Boeblingen
West Germany
(070 31) 2 30 14

Sydney (Australian H.Q.)
I.P. Sharp Associates Pty. Ltd.
8th Floor Carlton Centre
55 Elizabeth St.
Sydney, New South Wales 2000,
Australia
(02) 232-6366

Toronto
(International Headquarters)
I.P. Sharp Associates Limited
145 King Street West
Toronto, Ontario M5H 1J8
(416) 364-5361

Toronto (Special Systems Div.)
I.P. Sharp Associates Limited
156 Front Street W., 5th Floor
Toronto, Ontario M5J 1G6
(416) 364-5361

Vancouver
I.P. Sharp Associates Limited
Suite 902, 700 West Pender St.
Vancouver, B.C. V6C 1G8
(604) 687-8991

Victoria
I.P. Sharp Associates Limited
Chancery Court
1218 Langley Street
Victoria, B.C. V8W 1W2
(604) 388-6365

Vienna
I.P. Sharp Ges.mbH
Rechte Wienzeile 5/3
A-1040 Wien, Austria
(0222) 57 65 71

Washington
I.P. Sharp Associates, Inc.
Suite 305, 2033 K Street N.W.
Washington, D.C. 20006
(202) 293-2915

White Plains
I.P. Sharp Associates Inc.
Suite 312 West, 701 Westchester Avenue
White Plains, New York 10604, U.S.A.
(914) 328-8520

Winnipeg
I.P. Sharp Associates Limited
Suite 208
213 Notre Dame Avenue
Winnipeg, Manitoba R3B 1N3
(204) 947-1241

Zurich
I.P. Sharp A.G.
Fortunagasse 15
8001 Zurich
Switzerland
(01) 211 84 24

SHARP APL Communications Network

APL OPERATOR VOICE (416) 363-2051 **COMMUNICATIONS (416) 363-1832**

Local dial access is available in all locations listed above and in:

• Alliance • Ann Arbor • Austin • Baltimore • Birmingham • Buffalo • Clewiston (Fl) • Dayton • Des Moines • Des Plaines • Detroit • Ft. Lauderdale • Greenwich (Ct) • Halifax • Hartford • Hull • Knoxville • Laurel • Leeds • Liverpool • Lyndhurst • Minneapolis • New Orleans • Oxford • Quebec City • Raleigh • Red Deer • Regina • Rome • Santa Ana • Schwetzingen • Sunnyvale • Syracuse • Towanda • Ukiah • Warrington

Our private, packet-switched network connects with the Value Added Networks in:

• Alaska • Argentina • Bahrain • Bermuda • Finland • Hawaii • Israel • Luxembourg • Mexico • New Zealand • The Philippines • Portugal • Puerto Rico • Taiwan

In the continental United States the SHARP APL Network is interconnected with the Value Added networks to provide access in 170 more cities, and in Canada with 40 more. In all, with the 80 cities served by the I.P. Sharp Network listed above, SHARP APL is accessible from close to 300 places via a local phone call. Please ask at your nearest I.P. Sharp office for a complete list of access points and access procedures. Our private network also connects with the worldwide Telex network via the Rochester, N.Y. and Amsterdam codes.

technical supplement 34

APPLICATION OF COMPLEX NUMBERS TO SAMPLED DATA SYSTEMS

Helmut Mach, Philips Vienna, Austria

Abstract

The extension of primitive functions to complex numbers has greatly simplified computations in the field of system theory. This article will demonstrate some of the new capabilities which recently became available in SHARP APL. Two examples show that complex numbers are useful, not only in the technical area, but also in commercial applications in the analysis of time series.

For those who are not familiar with the subject, a brief introduction to the theory of sampled data systems, without involving mathematical details, follows. More on this topic appears in *Theory and Application of Digital Signal Processing*, L. Rabiner, B. Gold, Prentice Hall, 1975.

Theory

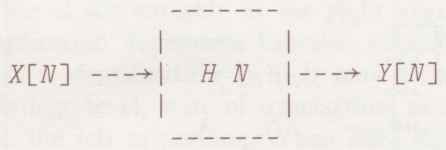


Figure 1

A sampled data system (Figure 1) takes input data `X[N]` at discrete points of time `N×T` (`T` is the sampling interval) and computes output values `Y[N]` according to an algorithm which is the solution of the difference equation of the system. As will be seen later, these calculations in the time domain are simple arithmetic operations with real numbers. Although the relations between `X[N]` and `Y[N]` in the time domain describe the system completely, there is an equal interest in the description in the frequency domain. This behaviour is conveniently characterized by means of the Z-transform. The so-called transfer function `H` is defined by

(1)

```
H Z ↔ ÷/(÷Z)⊥⊖AB
```

where `AB` is a matrix of two columns. `(1↑⍴AB)-1` is the order of the system. `Z` is a complex variable in the frequency domain.

For a second order system, straightforward manipulations and inverse transform (back into the time domain) yield the difference equation mentioned above:

(2)

```
Y[N] ↔ (AB[;0]+.×X[N-0 1 2]-
       (1↓AB[;1])+.×Y[N-1 2]
```

(Origin 0 is used throughout this article.) The frequency response is evaluated by putting `Z←*0J1×ω×T` so that

```
H *0J1×ω×T ↔ ÷/(÷0J1×ω×T)⊥⊖AB
           ↔ ÷/(¯12○-ω×T)⊥⊖AB
```

For the sake of completeness the structure of the system is given in figure 2.

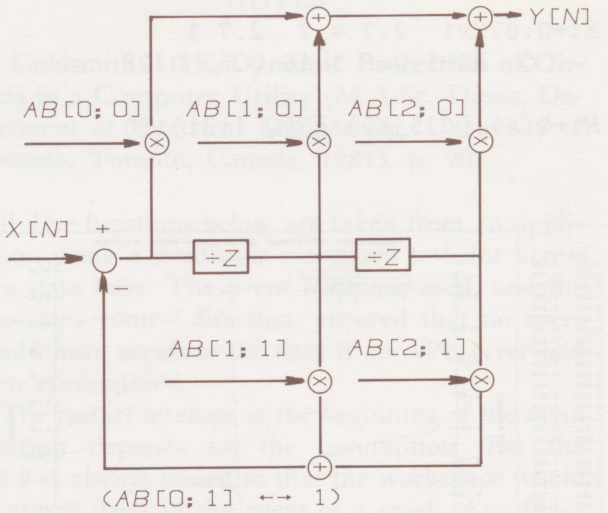


Figure 2

Software

The dyadic function *FREQ* takes as its left argument the matrix of coefficients, *AB*. The right argument is a vector of the normalized frequencies (not necessarily equally spaced) for which the response is to be calculated.

```
    ∇Z←AB FREQ W;CFM
[1]  ⍝ FREQUENCY RESPONSE OF A RECURSIVE FILTER.
[2]  ⍝AB←→MATRIX OF COEFFICIENTS    <H.MACH,PHILIPS>
[3]  ⍝W←→VECTOR OF NORMALIZED FREQUENCIES (0 TO 0.5)
[4]   CFM←¯1200W∘.×,¯2
[5]   Z← 10 12 ∘.○÷/CFM⊥⍉AB
      ∇
```

Applications

Since the evaluation of the coefficients for the specific system is not part of this article, we will assume that the data has been generated by other methods, such as those described in Rabiner-Gold.

Example 1: An electrical **high-pass filter** is given as a fourth order system by:

```
H Z ↔ ÷/(÷Z)⊥⍉⌽2 5⍴(0.072×1 ¯2.7
      4.7 ¯2.7 1),1 0.8 1.16 0.63 0.28
```

Now we execute function *FREQ*:

```
K1←0.072×1 ¯2.7 4.7 ¯2.7 1
    ◇ K2←1 0.8 1.16 0.63 0.28

M1←⌽(K1,[0.5]K2) FREQ (⍳31)÷60
```

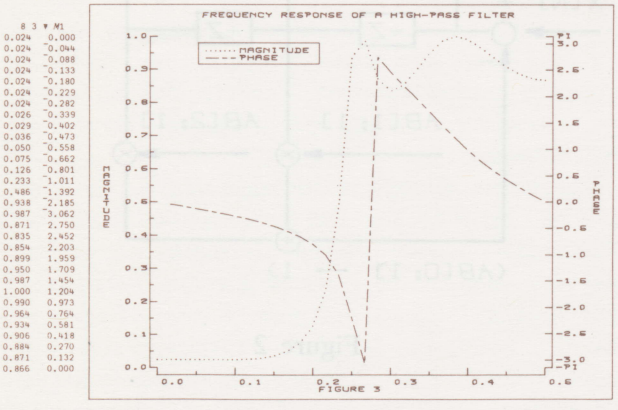


Figure 3
(plot obtained via SUPERPLOT).

Example 2: As a second example let us consider a commercial problem, the smoothing of a time series. Silver prices from the London Metal Exchange (LME) for the first 146 trading days of 1981 were taken from the Commodity data base. We want to analyze the series to see long term variations (greater than, say 30 days) but we are not interested in short term fluctuations (less than, say 3 days). We want to "filter out" these fluctuations. The system that performs the smoothing is actually a **low-pass-filter**. A second order system with the following transfer function will do quite well:

```
H Z ↔ ÷/(÷Z)⊥⍉⌽2 3⍴(0.068×
        1 2 1),1 ¯1.141 0.413
```

Again we use function *FREQ*:

```
K1←0.068×1 2 1  ◇  K2←1 ¯1.141 0.413

M2←⌽(K1,[0.5]K2) FREQ  (⍳31)÷60
```

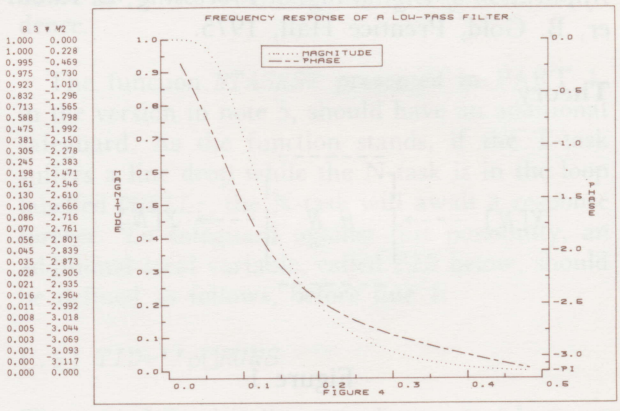


Figure 4

In order to accomplish filtering of the time series we apply equation (2) and get the result shown in figure 5, where both the raw and the smoothed data are plotted. The long term variations are quite clearly recognizable and no longer are obscured by the daily fluctuations. Note that this filter introduces a time delay. To take it into account for corrective action, you have to use the phase plot (figure 4) to estimate the delay.

The exact value of the delay can be calculated via function *T*:

```
T:
-(+/((αx-ιρα)
P ω)-(ωx-ιρω) P α)÷+/α P ω
```

where α is *K1* and ω is *K2* above.

P: +⌿(-ιρα)φα∘.xω,0x1↓α is a known function for the product of two polynomials.

```
      K1 T K2
2.158
```

The time delay is 2.158 days in our example.

Equation (2) was translated into a straightforward APL function *FILTER* displayed below. The coefficients of the X-terms and the Y-terms are globally specified as *A* and *B*.

```
A←0.068 x 1 2 1

B←¯1.141 0.413

    ∇ Z←FILTER ω;X;Y;YN;I
[1]   ⍝ FILTER SMOOTHS A TIME SERIES
                    <H.MACH,PHILIPS>
[2]   ⍝ A AND B GLOBAL COEFFICIENTS
[3]   ⍝ ω RAW TIME SERIES DATA
[4] ⍝ Z FILTERED (SMOOTHED) DATA
[5]   X←3/0 ◇ Y←2/0 ◇ I←0 ◇ Z←ι0
[6]   L1:→((¯3+ρω)<I←I+1)/END
[7]   YN←(-B+.xY)+A+.xX ⍝ EQUATION (2)
[8]   X←(1↑ω),¯1↓X ◇ Y←YN,¯1↓Y
[9]   ω←1φω ◇ Z←Z,YN ◇ →L1
[10] END:
    ∇
```

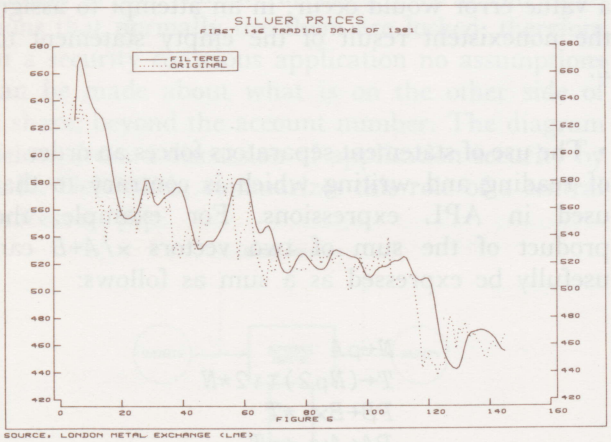


Figure 5

Conclusion

There is no doubt about the power and simplicity of calculations with complex numbers in technical applications. But, generally speaking, filters need not be restricted to technical problems. For commercial applications, a filter like the one above, although simple, offers considerable flexibility in the analysis of time series. It is a fast and inexpensive tool.

Acknowledgement: The encouragement and assistance of the I.P. Sharp representatives in Vienna and Toronto are highly appreciated.

COMMENT

Palo Alto, Sunday, 30 Aug 1981

The Contest 11 results were fascinating to read, and the winning entries were certainly masterpieces. However, I was startled to find in the text the statement that '⎕*UL* is not a primitive function'. The *SHARP APL Reference Manual* (p.14, in box) says "a function may be primitive (denoted by a symbol **or distinguished name**.... emphasis mine). More to the point, perhaps, the same manual (p.20) says, "SHARP APL includes a number of niladic primitive functions (⎕*AI*, ⎕*TS*, ⎕*NAMES*, ⎕*NUMS*, ⎕*AV*, ⎕*UL*, ⎕*WA*, and ⎕*LC*.)" It is perhaps debatable whether system functions are primitive, but, in a SHARP APL environment, in a context in which 'primitive' was unqualified, it seems most reasonable to abide by the definitions in the Reference Manual.

I'd like to express my appreciation for the statement of the rules and criteria in the report on the results.

Roland Pesch

London, Monday August 31, 1981

It doesn't look like the sort of area where reason necessarily prevails, but Paul Berry's definition of ⎕functions as primitives is at variance with colloquial usage (as exemplified by Iverson and Bernecky, *Operators and Arrays* in the proceedings of

the 1980 SHARP APL Users Meeting), at variance with Gilman and Rose, and at variance with Adin Falkoff's draft standard.

Phil Chastney

Hong Kong, Thursday September 24, 1981

I found paragraph 2, column 2 on page 4 of the August newsletter slightly misleading — my fault? Having just discussed close composition, I was expecting to read something like: "..Thus when the righthand function (say ⍵) of the composition operator has unlimited rank (in particular if⍤ then right-unlimited-rank; if⍥ then right and left unlimited-rank), the effect of composition is no different..."

It is when ⍵ is a limited-rank (e.g., **scalar**) function that the derived function thus created has properties you could not get in any other way... however, when the **lefthand** function (say ⍺) is scalar then ⍺⍤⍵ *Y* (or ⍺⍥⍵ *Y*) is equivalent to ⍺ ⍵ *Y*.

Arthur Whitney

STATEMENT SEPARATORS

Robert Bernecky and Kenneth E. Iverson,
Toronto

Standards Committees have taken an interest in the problem of statement separators. We support the position taken by the CSA that including a statement separator (◇) in the APL language standard is **not** desirable. This article attempts to clarify some of the reasons; comments are welcome, and should be addressed to the authors.

Proponents of statement separator offer several arguments in favour:

• Statement separators provide some measure of security, in that branching into the middle of a collection of separated statements is not permitted, and in the absence of defined functions, the collection will not be interrupted by a weak interrupt. For example, in

```
S1 ◇ S2 ◇ S3
```

if statement *S3* is executed, it is known that statements *S1* and *S2* were executed. A user may be able to interrupt *S1*, *S2*, or *S3*, but restarting the line will always begin with *S1*. The occurrence of a defined function within *S1*, *S2*, or *S3* might allow suspension of the function, and subsequent tinkering.

• Statement separators allow grouping of related APL statements on a single line.

• Separated statements use less paper, by virtue of allowing APL statements to spread across, as well as down, the page.

The arguments against a statement separator are threefold:

• A statement separator introduces a new syntactic element into APL: It is not a function, operator, array, or grouping symbol, and therefore requires its own set of rules. For example,a statement separator must interplay with ⍎, ⍎, and →, and some aspects of this are less than obvious. In general, the execute function produces the same effect as placing the text of the argument into the function line. When statement separator is introduced, this simple definition is lost.

Moreover (at least in SHARP APL) if *X*←'⍳5', the result of *Z*←⍎*X* is to assign ⍳5 to *Z*. However, if *X* contains statement separators, the effect is not so simple. If *X*←'⍳5 ◇ 666', ⍳5 will print on the terminal, and *Z* will be assigned the value 666. If a trailing statement separator was appended to *X*, ⍳5 and 666 would print on the terminal, then a value error would occur, in an attempt to assign the nonexistent result of the empty statement to *Z*.

• The use of statement separators forces an order of reading and writing which is contrary to that used in APL expressions. For example, the product of the sum of two vectors ×/*A*+*B* can usefully be expressed as a sum as follows:

```
N←⍴A
T←(N⍴2)⊤⍳2*N
PB←B×.*T
PA←A×.*~T
R←+/PA×PB
```

This would normally be converted to a 'diamondized' expression as follows:

```
N←⍴A ◇ T←(N⍴2)⊤⍳2*N
     ◇ PB←B⌿.*T
     ◇ PA←A⌿.*~T ◇ R←+/PA×PB
```

Written as a single APL expression without statement separators, this would normally be expressed (as on page 459 of *Notation as a Tool of Thought*) as follows:

```
R←+/(A⌿.*~T)×B⌿.*T←(N⍴2)⊤⍳2*N←⍴A
```

Besides having fewer symbols and temporary results, the latter expression has the usual advantage of reading analytically from left to right, and constructively from right to left. Because it is analytic from left to right, the overall result can be made to print by removing the *R*←. In the line using statement separators, the *R*← would have to be removed from the beginning of the last segment. Furthermore, the line containing statement separators cannot be read analytically or constructively, without zigzagging across the line.

It is the opinion of the authors that the benefits of statement separators might be better achieved in other ways, such as introduction of a LEV function which returns its left argument. *LEV*, being a normal APL function, would lead to no new rules.

NY SIGAPL — A Report

Robert Bernecky and R.C. Metzger from I.P. Sharp were invited to speak at the meeting of the ACM Special Interest Group for A Programming Language, on September 28th:

Bob Metzger from our Rochester office spoke about the advantages, applications and limitations of direct definition in his talk "Practical Applications of Direct Definition".

Bob Bernecky, from Toronto, discussed:
- An introduction to enclosed arrays, with simple examples chosen from real applications.
- Using a proposed extension to ⍕ with enclosed arrays to simplify report formatting (see p.T4).
- Composition operators and new idioms in APL.
- Coadunate representation.
- Similarities between enclosed arrays and APL files: can file systems be eliminated from APL?

Bob Bernecky

Bob was born and bred in Buffalo, NY, "schooled in the streets, Caltech, and SUNY Buffalo. My first computer experience was with an IBM 1620 and an IBM 1401 (rousing 8K bytes of memory), plus a few Burroughs tube machines (B220, B205). Since 1971, I've worked with the I.P. Sharp APL Development Group.

I live on Ward's Island in Toronto Harbor, loathe automobiles and their infrastructure, and ride a bicycle whenever possible, wherever possible."

A Model for the Extension of MONADIC THORN

Robert Bernecky, Toronto

The workspace 1 *SATN*41 contains a function *TH* which models a proposed extension to the definition of the monadic function ⍕⍵ as applied to enclosed arrays. *TH*, written by Peter Wooster after a slightly different version by Arthur Whitney, is a modified version of the *TH* function presented by Bernecky and Iverson [1], which is essentially the definition proposed by Jenkins and Michel [2]. The *TH* function presented here differs from the Jenkins-Michel definition in two respects: it is identical to ⍕ when applied to simple arrays, and the result preserves the shape of the argument in all but the last two elements of the shape vector when applied to enclosed arrays.

Description of the Function

The function ⍕ applies to an array ⍵ of numbers or characters by replacing each array element with the character array which represents that element. Since the character representation of any character or number occupies one row and one or more columns, the effect is to replace the 1-by-1 array element with a 1-by-N character array. Extra spaces are inserted between columns as necessary, to force the original columns to line up, and to separate columns of numbers.

The array formed by ⍕⍵ is, therefore, of the same shape as ⍵, except for the last element of the shape vector, where the character representation of the elements may have forced more columns into the array.

TH is identical to ⍕ when applied to simple (non-enclosed) arrays. When ⍵ is enclosed, *TH* ⍵ operates in a similar manner: it replaces each element of the array with the character array which represents that element, by applying *TH* recursively to each disclosed element.

Unlike character or numeric scalars which always format into 1 row, the character representation of a scalar which is an enclosed array may require an arbitrary number of rows and columns. Therefore, spaces are inserted between both rows and columns, to force the original rows and columns to line up with each other and to separate columns from one another. This means that the array *TH* ⍵ will be of the same shape as ⍵ in all but the last two elements of the shape vector, and that ⍴*TH* ⍵ when ⍵ is an enclosed array will always produce a result of rank 2 or greater.

When an element of ⍵ is enclosed more than once, *TH* is applied recursively until a non-enclosed array is encountered. The recursion then terminates, and the character arrays formed are catenated, with appropriate padding, to form the result at that level of recursion. *TH* always produces a rank 2 result when applied to a scalar which is enclosed, even when the result of applying ⍕ to the disclosed scalar would produce a result of higher rank. Extra rows of spaces are inserted into the result, so that the results print identically, even though the result ranks differ:

```
      A←2 3 4⍴⍳24
      ⍕A
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24

      ⍴TH A
2 3 11

      TH <A
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24

      ⍴TH <A
7 11
```

It is expected that the extended format function will provide a convenient way of formatting APL arrays, frequently replacing more complex expressions using the ⎕*FMT* system function. As will be seen in the two simple examples to follow, the proposed extension to ⍕ also provides several facilities which are not possible using ⎕*FMT* alone, and provides an inherent capability for aligning report title, body, and summary information.

Text Layout Example

Here we'll explore some possibilities of formatting two columns of text and a title:

```
      COL1←<2 9ρ'THIS ONE GOES LEFT'
COL2←<3
9ρ'AND THISGOES ONTHE RIGHT   '
      TH COL1,COL2
THIS ONE  AND THIS
GOES LEFT GOES ON
          THE RIGHT
```

So far so good. What about a title running across the top? "Top" suggests that a vector won't work any more, so let's make the array of shape 2 1, with the top row being the title, and the second row being the columnar text:

```
TITLE←<'THIS TITLE SPREADS ACROSS'
      TH 2 1ρTITLE,<COL1,COL2
THIS TITLE SPREADS ACROSS
THIS ONE  AND THIS
GOES LEFT GOES ON
          THE RIGHT
```

Note how we have taken advantage of the recursive nature of *TH* in the previous example. A two-page format might be achieved by:

```
    T←(<'LEFT PAGE TITLE'),<'RIGHT PAGE TITLE'
    TH 2 2ρT,COL1,COL2
LEFT PAGE TITLE RIGHT PAGE TITLE
THIS ONE        AND THIS
GOES LEFT       GOES ON
                THE RIGHT
```

Report Formatting Example

A typical business report consists of 3 basic elements: titles, report body, and totals or summary data. Herewith a simple weather report:

```
      CITIES
SEDRO WOOLEY
TORONTO
WARD'S ISLAND
TOTSUKA
HEIDELBERG
TIGER
      ρCITIES
6 13
      RAIN
0
0.5
0.1
1.2
      ρRAIN
4 1
      TEMPS
19
23
25
27.5
19
      ρTEMPS
5 1
      MEANS←(<'MEAN:'),(<(+/,TEMPS)÷ρ,TEMPS),<(+/,RAIN)÷ρ,RAIN
      TITLES←(<'LOCATION'),(<'TEMPERATURES'),<'RAIN'
      BODY←(<CITIES),(<TEMPS),<RAIN
      REPORT←TITLES,[⎕IO]BODY,[⎕IO-.5]MEANS
      TH REPORT
LOCATION      TEMPERATURES RAIN
SEDRO WOOLEY  19           0
TORONTO       23           0.5
WARD'S ISLAND 25           0.1
TOTSUKA       27.5         1.2
HEIDELBERG    19
TIGER
MEAN:         22.7         0.45
```

Note that the lack of data from Tiger and Heidelberg does not cause an error, and that the appropriate columns of title, body, and means line up without any need to count spaces, or to use a forms layout sheet.

Horizontal spacing of columns can be controlled by inserting an entire column of space at the desired location:

```
TH
REPORT[;1],(<'
'),REPORT[;2 3]
LOCATION      TEMPERATURES RAIN
SEDRO WOOLEY  19           0
TORONTO       23           0.5
WARD'S ISLAND 25           0.1
TOTSUKA       27.5         1.2
HEIDELBERG    19
TIGER
MEAN:         22.7         0.45
```

Another way to control horizontal spacing without otherwise affecting the result is to insert a column of appropriately shaped empty arrays:

```
      TH (<0 4⍴0),REPORT
   LOCATION      TEMPERATURES RAIN
   SEDRO WOOLEY  19           0
   TORONTO       23           0.5

WARD'S ISLAND 25           0.1

TOTSUKA       27.5         1.2
   HEIDELBERG    19
   TIGER
   MEAN:         22.7         0.45
```

It should be clear that empty arrays can be used in conjunction with the *TH* function in much the same way as the *X* qualifier is used in ⎕*FMT*.

As another example of how to get something for nothing, note that empty arrays can also be used to control vertical spacing:

```
TH (REPORT,[⎕IO]<2 0⍴0)[1 4 2 4 3;]
LOCATION      TEMPERATURES RAIN

SEDRO WOOLEY  19           0
TORONTO       23           0.5
WARD'S ISLAND 25           0.1
TOTSUKA       27.5         1.2
HEIDELBERG    19
TIGER

MEAN:         22.7         0.45
```

Column counting when altering the format of a report is one of the more boring, tedious tasks that an applications programmer is called upon to do. The *TH* function handles the common cases automatically. Let's change one of the titles and see what happens:

```
      REPORT[1;2]←<'TEMPS'
      TH REPORT
   LOCATION      TEMPS RAIN
   SEDRO WOOLEY  19    0
   TORONTO       23    0.5
   WARD'S ISLAND 25    0.1
   TOTSUKA       27.5  1.2
   HEIDELBERG    19
   TIGER
   MEAN:         22.7  0.45
```

Note that the remainder of the titles, the report body, and the means moved over to fill in the space introduced by changing the title.

The *TH* function is intended to serve two purposes: first, it provides users of enclosed arrays with a needed tool, until a primitive formatting function becomes available. Second, feedback from the user community is expected to give the APL development group an indication as to the utility of the proposed definition.

TH is not intended, by itself, to provide all formatting capabilities, and should be used with other APL functions to perform customized formatting. For example, the integer formatting of rainfall could be forced by executing either of the following APL statements before building the report array:

```
RAIN←⌊RAIN
RAIN←2 0⍕RAIN
```

More complex formatting capabilities are frequently achievable by use of an expression such as:

```
TH WD ⍕⍤2> RPT
```

Assuming that the extension modelled by *TH* is generally acceptable to the APL community, it will appear in a future release of SHARP APL as an extended definition of monadic ⍕, and as the default display for enclosed arrays. The *TH* model differs from the Whitney function in its ability to use a global variable, *QPP* (**p**rint **p**arameter), to control formatting. *QPP* is a three-element integer vector that corresponds to ⎕*PP*. The first element is used as ⎕*PP*. The second and third elements control the number of blank rows to be inserted between the rows and columns of the result. The default value of *QPP* is `10 0 1`. The utility of this extension is still to be evaluated.

References

[1] Bernecky, R., Iverson, K.E., *Operators and Enclosed Arrays,* I.P. Sharp Users Meeting, 1980.

[1] Jenkins, M.A., Michel, J., *Operators in an APL Including Nested Arrays,* Queen's University Technical Report 78-60 1978.

SHARED VARIABLES PART 2

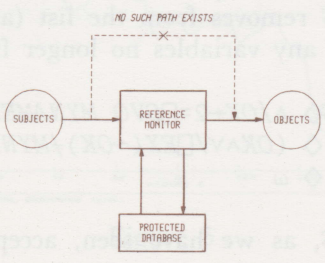
Roland Pesch, Palo Alto

In part 1, we discussed the use of shared variables in applications where the connected task serves one user.

2) Central Task with Many Users

The other major application of shared variables is a central task that serves many users. For example, a single central processor can mediate all access to the data. Most of the advantages mentioned for single-user connected tasks apply (though note that the central processor may itself create CTs for its own use); in addition, there are advantages to do with security and data integrity.

Security: first and perhaps foremost, if no functions in the application workspace directly access files, there is no way of finding out any passnumbers that may be required for access to the files. No functions in the user's workspace need be locked. The same is true of the CT workspace: as the user doesn't know its name, all of its functions can be left unlocked as well. In addition, much more detailed access control is possible than that provided by *⎕STAC* and *⎕HOLD*. Different users may be given different priorities; they can be allowed to read some information, but not all (creating different 'views' of the data base); they can be allowed to write some kinds of things, but not others. Two things, in particular, should be borne in mind when designing the central processor: (1) do all security checking in the central processor, and (2) don't trust the user's workspace. In the application workspace, the user can always take control of the shared variables, even if the functions that normally use them are locked; therefore in a security conscious application no assumptions can be made about what is on the other side of a share, beyond the account number. The diagram below, from a discussion of application security by L.H. Goldsmith, summarizes this role of a central processor: [9]



The central processor ("reference monitor") is the only path between users and the objects contained in the data base; as an intelligent intermediary, it consults a protected data base that only it has access to, to determine what users may view what data.

Data integrity can also benefit from the use of such a central processor to verify and write all data entered into a data base. When many users have the ability to update related or identical parts of a data base, there is always the possibility that something may go wrong with the software. This can be the source of an incredible mess if the error leaves a partly updated data base with some resulting inconsistencies. Other users can unknowingly damage the data base further by (apparently) normal operations.

A central processor is no less prone to software errors than anything else. There is, however, one crucial difference: all users are using the same copy of the software, so the software can refuse access after it has encountered an APL error. Chances are the error will be brought to a programmer's attention more quickly this way, too...[10]

All central processors share certain tasks, so it is perhaps inevitable that, at the outermost level, they should bear a marked resemblance to one another. A good example is P.C. Berry's *MONITOR* function, published in the *SHARP APL Reference Manual* (as amended in the *Additions and Corrections*) [11]. We will examine a version of it rewritten in direct definition to emphasize its nature as a continuous process. The monitor is invoked by *⎕LX* as follows:

```
⍎(0≠1↑3↓,⎕RUNS)/'MONITOR 99'
```

Its argument is the clone ID it will use. Note that for this sort of application, the clone ID must be fixed so that users who may not themselves have started the monitor can access it.

```
MONITOR◇ ⍵≠⎕SVN ⍵
       ◇ CONTROL FIRSTOFFERS
       ◇ ⎕SIGNAL 555
```

The monitor verifies that it is the authentic copy (that no other version of itself is running); this is necessary for any central processor that does not run permanently, as two users may attempt to start it simultaneously. If it concludes it's the

authorized version, it passes control to the function *CONTROL*, giving it as an argument the names and identifications of the initial offers to the processor, as determined by *FIRSTOFFERS*:

```
FIRSTOFFERS◇ 0∈ρX←⎕SVQ⍳0
           ◇ X ONEBY1 NAMES[⍳0;]
           ◇ NEWOFFERS NAMES[0ρ⎕DL AWHILE;]
```

As a user may have started the monitor either before or after making a share offer, *FIRSTOFFERS* takes one of two courses of action: if it already has offers, it collects their names and accepts them — otherwise it waits a while and proceeds to its usual means of collecting new offers,

```
NEWOFFERS◇ 0∈ρX←⎕SVQ⍳0
         ◇ X ONEBY1 ⍵
         ◇ ⍵
```

which, like *FIRSTOFFERS*, uses *ONEBY*1 for the main part of its work.

```
ONEBY1◇ 0∈ρα
      ◇ (1 0↓α) ONEBY1 ⍵ ON (1 2↑α) NEW1 ⍵
      ◇ ⍵
```

*ONEBY*1's name expresses its purpose: to inspect the share offers to the processor one by one (required by ⎕*SVQ*). *ON* is a utility sometimes called *CATENATEROWS*, which is what it does. *NEW*1 below

```
NEW1◇ (α FSVAR T),' ',(1↑ρT←α NAMEFN MYNAMES ⍵)≠⍳⍴α
```

deals with each offer. It accepts all share offers (through *FSVAR*) from the user named in its left argument, choosing for each a local name (through *NAMEFN*) distinct from those already in use (*MYNAMES* ⍵). At first, of course, there are no names in use (⍵ is empty). The result of *NEW*1 reflects the structure in which *MONITOR* keeps track of things: the local name of each shared variable (returned by *FSVAR* after establishing the share) followed by the identification of its corresponding user.

```
MYNAMES◇ ⍵[;⍳¯1↑ρNAMES]
```

simply extracts the names from such a structure. *NAMEFN*, in this example,

```
NAMEFN◇ NAMES[(1↑ρSURROG)↑
      C/⍳ρC←∧/NAMES∨.≠⍵;],' '
      ,SURROG←⎕SVQ ,α
```

picks its names from a list; one could generate them dynamically instead. In any event, *NAMEFN* places the names offered by the user (obtained from ⎕*SVQ*) in the position that will correspond to the user identification in the data structure we have seen; after *FSVAR* uses this information to establish the share, only the local name is required, so *FSVAR* discards the surrogate when it's done. *FSVAR* itself is quite straightforward:

```
FSVAR◇ ×ρ((1↑ρ⍵)≠α) ⎕SVO ⍵
     ◇
     ◇ MYNAMES(∨/0 0 1 1 ⎕SVC ⍵)≠⍵
```

It accepts the share offers, establishes an access control vector preventing the user from doing anything silly, and returns the (local) names it has just shared.

The *CONTROL* function expresses the major recursion of the monitor:

```
CONTROL◇ 0∈ρ⍵
       ◇ CONTROL PROC ⍵
       ◇ ''
```

As long as it has full shares (represented by entries in its argument) it continues executing. *PROC* represents one pass through this recursion:

```
PROC◇ ⎕SC
    ◇
    ◇ T WORK (⎕SVS MYNAMES T←
      NEWOFFERS OLDOFFERS
      ⍵)[;1+⎕IO]
```

PROC first waits in ⎕*SC* for something to happen (note that this, taken in conjunction with the recursion in *CONTROL*, constitutes a dwell); when it does, it proceeds to do whatever *WORK* is required, after updating its list of offers. *OLDOFFERS* removes from the list (and from the workspace) any variables no longer fully shared:

```
OLDOFFERS◇ ∧/OK←2=⎕SVO MYNAMES ⍵
         ◇ (OK∧∨/⎕EX(~OK)≠MYNAMES ⍵)≠⍵
         ◇ ⍵
```

NEWOFFERS, as we have seen, accepts any new

share offers and adds the corresponding entries to the list.

The updated list is passed to *WORK* as its left argument; *WORK* will return the list as the result when it's done. The right argument, the second column of the result of □*SVS*, contains a 1 for any variable that has been set by a user since the last pass, and hence determines what is to be worked on:

```
WORK◇ ∨/ω
    ◇ α
    ◇ α WORK ω>D,0ρDO ,(D←<\ω)≠α
```

WORK proceeds to call itself until all outstanding requests have been served (represented by all 0s in its right argument). At each pass, it selects the first entry in its right argument (<\ω), passes it to *DO* to take the action indicated, and marks that user as no longer waiting before going on to the next.

```
DO◇ ⍝ T,'←',(D↓ω),' PROCESS ',T←(D←1↓ρNAMES)↑ω
```

DO is the final function independent of the application: its argument contains the name and user identification of a request for action. It passes the value of the variable as the right argument to the application dependent function *PROCESS*, and the user's identification (which may be required for privilege level, state of transaction, accounting, etc.) as the left argument. When *PROCESS* comes up with a reply, it is assigned to the user's shared variable, and the next user can be served.

Most of what goes on in a function like *PROCESS* is of no concern to a discussion of shared variables. There is, however, one concern that may be quite frequent: if whatever it is that *PROCESS* does for one user takes too long, other users (who may have less time-consuming, but perhaps more urgent, things to do) may suffer. M. Symes has solved this problem, in writing a system task that provides scheduling services to APL users. He organized the application functions into pieces or 'quanta' of execution, giving them the ability to execute only part of a user's request at a time. Using such a scheme, *PROCESS* keeps track of the state each user's task is in, but performs only one quantum at each request (the user would have to keep requesting service until either a predetermined number of quanta had elapsed, or (s)he received some sort of signal that the process was complete. *PROCESS*, for its part, might have to verify that the proper sequence was maintained, and should be prepared to abort any sensitive action in progress if the user failed to respond properly). The quanta might be represented as a set of related functions, or they might be a set of possible branch points inside a single larger function.

A variation of this technique is possible: *PROCESS* might spawn further connected tasks for time-consuming requests. The initial response to the user would indicate 'started', and the monitor could go on to other requests; the user's side of the exchange would essentially be the repeated question 'done yet?' which *PROCESS* could respond to immediately by checking the state of the shared variable link to the secondary process.

Acknowledgements

I would like to thank Gottfried Bach, Paul Berry, Caille Colburn, Simon Garland, Leslie Goldsmith, Paul Jackson, Dick Lathwell, Gene McDonnell, Mark Seltzer, Jeff Smith, Mike Symes, and Joey Tuttle for their contributions to this article.

NOTES

[9] Goldsmith, L.H., **Dynamic Protection of Objects in a Computer Utility** (M.A.Sc. Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Canada, 1981), p. 90

[10] The functions below are taken from an application using a multiuser associated task for access to a data base. The event trapping used, and the secondary control function, ensured that no users would have access to the data if an APL error had been encountered.

The restart attempt at the beginning of the main function depends on the assumption that the □*RUN* is always issued so that the workspace would save over itself in the event of a crash. Any functions starting with Δ*R* are part of a set of restartable file functions. Δ*OFFERS* corresponds to the sequence *OLDOFFERS NEWOFFERS* in the example of a monitor in the text; Δ*PENDING* selects requests to answer, ordering them by priority.

It may also be of interest that the distinction between three different kinds of requests is encoded as part of the name offered to share.

```
      ∇ CONTROL FILEID; ANS; NEXT; SVARS; T; ⎕LX; CLONE; ⎕ER
    ; ΔFNM; ⎕TRAP
[1]       ⍝ CONTROL FN IN DATA-ACCESS NTASK.  CALLED BY ⎕LX.
[2]       ⍝ <FILEID> IS NAME OF FILE CONTAINING DATA BASE.
[3]       ⍝ SETS <ΔFNM, LOCAL GLOBAL FOR FNS CALLED BY THIS ONE.
[4]       ⍝          ROPE 7 MAY 1979.
[5]       ⎕LX←'→(0=⎕RUNS[1;4])↑0 ◇ CR←ΔRESTART ◇ SVARS←CRASHED ''''
    ◇ →CR' ⍝ IF ERROR EVENT, ISSUE WARNINGS TO USERS, REFUSE DATA.
    OTHERWISE RESTART.
[6]       ⎕TRAP←'∘0 E ERR←⎕ER ◇ ⍎NEXT[1;],''←CRSHMSG''
    ◇ CRASHED SVARS'
[7]       →(~1=CLONE←⎕SVN 99)↑AWAY          ⍝ GO AWAY IF CLONE ID IS
    TAKEN.
[8]       ΔRINIT
[9]       FILEID ΔRSTIE ΔFNM←ΔRNUM,28 ◇ SVARS←ΔOFFERS
[10] CHECK: T←⎕SC
[11]      →(2∈⎕SVO SVARS)↑AWAY
[12]      NEXT←ΔPENDING SVARS
[13] LOOP:→(0∈⍴NEXT)↑CHECK
[14]      →(RDD,WRD,REF)['DWR'⍳1↑NEXT[1;]]
    ⍝ ALL SHARED VARS IN THIS SYS START WITH D, W, OR R
[15]  RDD: ANS←ΔGETDATA NEXT[1;] ◇ →BACK
[16]  WRD: ANS←ΔPUTDATA NEXT[1;] ◇ →BACK
[17]  REF: ANS←ΔREFERENCE NEXT[1;]
[18] BACK: ⍎NEXT[1;],'←ANS'
[19]      NEXT←1 0↓NEXT ◇ →LOOP
[20] AWAY: CLEAROUT
      ∇

     ∇ SV←CRASHED SV; ⎕ER; CLN; NEXT; T; ⎕TRAP
[1]       ⍝ NTASK CRASH RECOVERY OR WARNING MONITOR; CALLED BY ⎕LX OF
    CRASHED D'BASE RETRIEVAL NTASK.
[2]       ⍝ <SV>, MATRIX OF ALL VARIABLES SHARED, RETURNED IFF CRASH
    NOT CAUSED BY ERROR EVENT.
[3]       ⍝ USES GLOBAL CRSHMSG, WARNING CRASH MESSAGE.
[4]       ⍝ ROPE 7 MAY 1979.
[5]       ⎕TRAP←'∘0 1000 S'
[6]       →(~1=CLN←⎕SVN 99)↑AWAY
[7]       →(0∈⍴SV)↑CRI
[8]       SV←ΔOFFERS
[9]  CRI:→(1000>1↑⎕FI ERR[1;])↑OUT ⍝ ERR IS COPY OF OUTER ⎕ER SET UP
    BY ⎕TRAP.
[10] CHK: T←⎕SC
[11]      →(2∈⎕SVO SV)↑AWAY
[12]      NEXT←ΔPENDING SV
[13] LOOP:→(0∈⍴NEXT)↑CHK
[14]      ⍎NEXT[1;],'←CRSHMSG'
[15]      NEXT←1 0↓NEXT ◇ →LOOP
[16] AWAY:→                          ⍝ NO <CLEAROUT> TO ENSURE ERROR
     PRESERVED (MAY BE CALLED DIRECTLY BY DYING NTASK)
[17] OUT: ∇
```

[11] Berry, P.C., **SHARP APL Reference Manual** (I.P. Sharp Associates Ltd., March 1979), p. 97-98.

TERMINOLOGY, CORRECTIONS, and NOTATION

PART 1 of this article used the term 'auxiliary processor' to refer to an APL task whose major purpose is to collaborate with another APL task. Observant readers will have noted that, in the same issue of the newsletter, an auxiliary processor was defined as a non-APL program. I had meant to use the term 'associated processor', to simultaneously highlight the similarity with the other kind of AP, and yet use a different term in recognition of the standard definition of 'auxiliary processor'. In hindsight, the confusion between AP as I used it and AP as it was generally used seems quite reasonable: accordingly, in PART 2, I have abandoned the earlier terminology and refer to these tasks as 'connected tasks'.

My apologies to the X3J10 standards committee: I was wrong to choose their discussions as an example of the confusion between shared variables and files. While the confusion exists, the standards committee is innocent of it; my impression of their early discussion on the subject was erroneous and based on a too-hasty reading of their correspondence.

The function *NTASKSV* presented in PART 1, in the version in note 5, should have an additional safeguard. As the function stands, if the T-task suffers a line drop while the N-task is in the loop labelled *DWELL*:, the N-task will await a response forever. To safeguard against this possibility, an additional local variable, called *TID* below, should be defined as follows, before line 1:

```
[.5] TID←''⍴⎕RUNS
```

Then the following line may be inserted between [10] and [11] to make the function exit if the T-task is gone:

```
[10.5] →(TID∈⎕RUNS[;⎕IO])↓0
```

My thanks to Constandinos Appla for pointing this out.

The notation used for direct definition of functions in both parts of this article differs from the traditional one, and is that used in the workspace 12 *COURSE*. There are two major differences: the statement separator is ◇, and, in the conditional form, the selector statement appears first, rather than between the two other statements.